<CODEWARS2026/>

BCN ▪ LEO ▪ VAL

Barcelona 2026

Problems and solutions

#	Problem	Points
1	Nether Portal	2
2	The Git Commit Counter	2
3	Fallas Budget Calculator	2
4	Robo-Ref The Offside Line	3
5	Even Odd Cinema	3
6	Highest Digit	4
7	PrincessIntervalla	5
8	Sheep Counter	5
9	Parking Spot Finder	5
10	Code Warrior	5
11	Count Peaks	7
12	Garden Planner	7
13	The Time Rebels Of France	7
14	Spam Filter	7
15	Palindrome Maker	9
16	From One Chord To Another	9
17	Alphabet Rhomboid	10
18	Laser Gate Alignment	11
19	Oil Well Exploration	12
20	Find the Unique Pair	13
21	Ghost In The Static	13
22	Bullseye	13
23	Molecular Mass Puzzle	14
24	Pizza Delivery Disaster	15
25	NBA Dynamic Plus Minus Tracking	15
26	Broken Calculator (Interactive)	16
27	Parallel 3D Printing	17
28	Largest Island	18
29	Painting Robot Assignment	22
30	Shiritori Chain	25
31	The Counterfeit Coin (Interactive)	35

1 Nether Portal

2 points

Introduction

In Minecraft, the Nether is scaled down by a factor of 8 relative to the Overworld for the horizontal axes. This means that moving 1 block in the Nether corresponds to moving 8 blocks in the Overworld. Players exploit this by building "Nether highways" (long straight tunnels, often with packed ice for boats) to reach distant biomes, player bases, strongholds, and resource farms much faster than traveling in the Overworld.

To link portals correctly you must convert Overworld coordinates to Nether coordinates. A mistake can cause a portal to link to the wrong destination or spawn an unwanted portal. This challenge simulates the core arithmetic behind planning a highway or portal hub.

Imagine you are mapping a server-wide highway hub. You receive an Overworld base location and need to post the matching Nether waypoint so builders can extend the ice road. Quick, accurate math keeps travel networks tidy and prevents portal clutter. Only x and z are scaled by 8. The y (vertical) coordinate remains unchanged. Given Overworld coordinates (x, y, z), compute the corresponding Nether coordinates (x', y', z').

Input

Three lines where first line has an integer as x coordinate, second line has an integer as y coordinate and third line has an integer as z coordinate.

Output

A single line with three integers separated by spaces.

Example

Input

```
80
64
160
```

Output

```
10 64 20
```

Python

```python
def convert_overworld_to_nether(x: int, y: int, z: int):
    """Return Nether coordinates (x', y', z') using floor division for x,z.
    For typical Minecraft portal linking, overworld x,z should be multiples of
8.
    """
    return x // 8, y, z // 8


def main():
    x = int(input())
    y = int(input())
    z = int(input())

    nx, ny, nz = convert_overworld_to_nether(x, y, z)
    print(nx, ny, nz)


if __name__ == "__main__":
    main()
```

2 The Git Commit Counter

2 points

Introduction

You're building a developer dashboard that displays statistics from your team's Git repository. One of the features shows how long ago commits were made, displaying messages like "Last commit: 5 minutes ago" or "Last commit: 1 minute ago."

Here's the problem: nothing screams "amateur developer" louder than bad grammar in your app. You've seen it before - "1 minutes ago" makes even the best app look unprofessional. Your users are developers, and developers notice these things. They'll judge your entire codebase based on this one tiny detail.

Your task is to write a program that generates these timestamp messages with perfect grammar. The rule is simple: only the number 1 gets the singular form "minute". Everything else - including zero - uses the plural "minutes". (Yes, "0 minutes ago" is grammatically correct because zero is plural in English.)

Input

A single line containing an integer from 0 to 99 (inclusive)

Output

Print the following phrase:

```
Last commit: N minute(s) ago
```

where "minute" is singular only when N is 1. Use "minutes" (plural) for 0 and all numbers greater than 1. Finally replace N with the actual number from the input.

Example

Input

```
5
```

Output

```
Last commit: 5 minutes ago
```

Python

```python
n = int(input())

if n == 1:
    print(f"Last commit: {n} minute ago")
else:
    print(f"Last commit: {n} minutes ago")
```

3 Fallas Budget Calculator

2 points

Introduction

You plan to travel from Barcelona to Valencia to experience the famous Las Fallas. You found a hotel that offers special packages for visitors during the festivities.

The hotel's pricing system works as follows:

- The first night is charged at full price
- Every night after the first one has a fixed 10% discount off the standard price

How much will you spend on accommodation?

Input

The input contains two integers on separate lines:

- First line: the number of nights you will stay ($2 \leq \text{nights} \leq 100$)
- Second line: the price per night in euros (must be divisible by 10, so $10 \leq \text{price} \leq 1000$)

Output

Print the total cost for the accommodation as a rounded integer.

Example

Input

```
3
80
```

Output

```
224
```

Python

```python
# Read input
nights = int(input())
price = int(input())

# Additional nights with 10% discount
discounted_price = price * 9 // 10  # Integer division for whole number
additional_nights = nights - 1
total_cost = price + (discounted_price * additional_nights)

# Output result as integer
print(total_cost)
```

4 Robo-Ref The Offside Line

3 points

Introduction

In the year 2026, human referees have been replaced by HP AI systems. Your task is to program the module that detects offsides. In this simplified version of soccer, the field is a 2D plane. The attacking team is moving towards the goal located at X = 100.

A player is OFFSIDE if, at the moment the ball is played both conditions are true:

1. Its X-coordinate is greater (closer to goal) than Ball's X-coordinate.
2. Its X-coordinate is greater (closer to goal) than Second-Last Opponent's X-coordinate.

> ***Note:*** *Assume the Goalkeeper is always the "Last Opponent" at X=100, so we only care about the "Second-Last Opponent" (the last defender).*

Given the coordinates of the Attacker, the Ball, and the Last Defender, determine if the goal is valid or offside.

Input

Three lines, each containing two positive integers representing (x, y) coordinates:

- First line represents Attacker Position (A_x, A_y)
- Second line represents Ball Position (B_x, B_y)
- Third line represents Defender Position (D_x, D_y)

Output

Print OFFSIDE if the rules are violated or print GOAL if the position is valid.

Example 1

Input

```
70 25
80 30
85 20
```

Output

```
GOAL
```

Example 2

Input

```
90 25
80 30
85 20
```

Output

```
OFFSIDE
```

Python

```python
# Read attacker position
ax, ay = map(int, input().split())

# Read ball position
bx, by = map(int, input().split())

# Read defender position
dx, dy = map(int, input().split())

# Offside Logic
# Rule 1: Attacker must be ahead of the ball (ax > bx)
# Rule 2: Attacker must be ahead of the defender (ax > dx)

if ax > bx and ax > dx:
    print("OFFSIDE")
else:
    print("GOAL")
```

5 The Even And Odd Cinema

3 points

Introduction

In the town of Algorithmville, there exists a unique movie theater complex called "The Even and Odd Cinema" with three separate screens. When you buy a ticket, it has a positive integer printed on it, and here's how it works:

If the integer is between 1 and 9 (inclusive), an usher recites its English written word (e.g., "three" for 3), and you will enter the main movie theater to be surprised by the premiere film of the week. However, if the integer is greater than 9:

- If it's even, the usher will redirect you to the "Even Auditorium" to watch a funny comedy movie.
- If it's odd, the usher, with a smile, will open the door to the "Odd Theater" and you will experience a terror film.

Unfortunately, the usher is ill this weekend. To carry out his duties, the cinema has a computer. Can you write a program that accommodates the viewers?

Input

A single line containing the positive integer printed on the ticket.

Output

Print the appropriate English representation of the given number, "even", or "odd", based on the rules described in the introduction.

Example 1

Input

```
3
```

Output

```
three
```

Example 2

Input

```
11
```

Output

```
odd
```

Example 3

Input

```
20
```

Output

```
even
```

Python

```python
ticket = int(input())

table = ["one", "two", "three", "four", "five", "six", "seven", "eight", "nine"]

if ticket <= 9:
    print(table[ticket-1])
else:
    if ticket % 2 == 0:
        print("even")
    else:
        print("odd")
```

6 Highest Digit

4 points

Introduction

Your friend Alex has invented a quirky game called "The Lucky Number Game" for their birthday party. The rules are simple but fun: each guest receives a ticket with a number on it, and the "lucky digit" of that ticket is the highest digit in the number.

For example, if you get ticket number 379, your lucky digit is 9. If someone gets ticket 2, their lucky digit is simply 2. The person with the highest lucky digit at the party wins a prize!

Alex's party has 100 guests, and they need help quickly determining everyone's lucky digit. Can you write a program to find the highest digit in any given number?

Input

A single positive integer N ($1 \leq N \leq 10^9$)

Output

Print a single integer: the highest digit that appears in N.

Example 1

Input

```
379
```

Output

```
9
```

Example 2

Input

```
2
```

Output

```
2
```

Example 3

Input

```
377401
```

Output

```
7
```

Example 4

Input

```
1000000
```

Output

```
1
```

Python

```python
def highest_digit(n):
    """
    Find the highest digit in a positive integer.

    Args:
        n: A positive integer

    Returns:
        The highest digit (0-9) found in n
    """
    # Convert to string and find the maximum character
    max = -1
    for i in str(n):
        if int(i) > max:
            max = int(i)
    return max

def main():
    n = int(input())
    result = highest_digit(n)
    print(result)

if __name__ == "__main__":
    main()
```

7 Princess Intervalla's Number Drama

5 points

Introduction

Princess Intervalla is obsessed with her daily number collection. She always checks how dramatic it is – by searching for the interval (biggest number minus smallest) in the collection itself. She believes that the right interval brings good luck to the kingdom, so she checks if it's already in her list. If it is, her royal mood improves and blessings are guaranteed! So, if the interval is in the list, she smiles :) If not, it's a total flop :(

E.g. given the number collection 21, 35, 7, 14, 42, 28, 49 the interval is 49 - 7 = 42. Then a smile should be printed since 42 can be found in the list.

Input

It consists of several lines representing the daily number collection of Princess Intervalla. The first line contains the total number of elements in the collection, followed by that many lines, each containing a single positive number.

Output

Print a smile:) if the interval is in the list. Otherwise print :(

Example 1

Input

```
7
21
35
7
14
42
28
49
```

Output

```
:)
```

Example 2

Input

```
2
23
11
```

Output

```
:(
```

C++

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

int main() {
    // Read total number of elements
    int n;
    cin >> n;

    // Read n numbers and store them in a vector
    vector<int> numbers(n);
    for (auto& num : numbers) {
        cin >> num;
    }

    // Calculate the interval: max - min (single pass with minmax_element - C++11)
    auto minmax = minmax_element(numbers.begin(), numbers.end());
    int interval = *minmax.second - *minmax.first;

    // Check if the interval is in the list
    if (find(numbers.begin(), numbers.end(), interval) != numbers.end()) {
        cout << ":)" << endl;
    } else {
        cout << ":(" << endl;
    }

    return 0;
}
```

8 Sheep Counter

5 points

Introduction

Farmer Bob has a peculiar problem. Every night before bed, he counts his sheep to fall asleep. But Bob is terrible at counting - he always skips some numbers! For example, he might count "1, 2, 3, 8, 9, 10" completely forgetting about sheep 4, 5, 6, and 7.

Bob's therapist suggested he calculate the sum of all the sheep numbers he forgot to count each night. This way, Bob will realize how bad he is at counting and hopefully improve. The problem is, Bob is also terrible at addition!

Help Bob by writing a program that takes the sheep numbers he actually counted and calculates the sum of all the missing sheep numbers in his counting range.

Important! Bob is bad at counting, but at least he never forgets the lowest number and highest number sheep in the sequence, but he might skip some in between. This means that the counting range is strictly defined by the smallest and largest numbers Bob counted, but he might count sheep in any random order (he's really bad at this). At least Bob guarantees he won't count the same sheep twice (at least he got that right!). If all numbers in the range are present, the sum of missing numbers is 0.

Input

A single line containing a list of integers representing the sheep numbers Bob counted. The numbers are space-separated and can be in any order. The list will contain at least 2 numbers, and all numbers will be positive integers.

Output

Print a single integer: the sum of all missing sheep numbers in Bob's counting range. If Bob didn't skip any numbers (congratulations Bob!), print 0.

Example 1

Input

```
4 3 8 1 2
```

Output

```
18
```

Example 2

Input

```
17 16 15 10 11 12
```

Output

```
27
```

Example 3

Input

```
1 2 3 4 5
```

Output

```
0
```

Example 4

Input

```
50 45
```

Output

```
190
```

Python

```python
# Read the sheep numbers Bob counted
input_numbers = input().split()
numbers = []
for i in input_numbers:
    numbers.append(int(i))

# Find the range of Bob's counting
min_num = min(numbers)
max_num = max(numbers)

# Calculate sum of all numbers in the range [min, max]
# Using formula: sum = n * (first + last) / 2
# where n is the count of numbers
expected_sum = (max_num - min_num + 1) * (min_num + max_num) // 2

# Calculate sum of numbers Bob actually counted
actual_sum = sum(numbers)

# The missing sum is the difference
missing_sum = expected_sum - actual_sum

print(missing_sum)
```

9 Parking Spot Finder

5 points

Introduction

You need to help a driver find a parking spot on a busy street! The street is represented as a string where different characters indicate occupied or empty spaces. Your task is to determine if a car of a given size can fit in the empty spots available, and if so, mark where the car will park.

The car needs a continuous sequence of empty spaces (represented by `_`) to park. If such a space exists, you should replace those empty spots with asterisks (`*`) to show where the car is parked. The car will always park in the first available spot found when scanning the street from left to right.

If no suitable parking spot exists (because all empty spaces are too small or occupied), return the original street unchanged.

Input

The input consists of two lines:
First line: An integer representing the size of the car (number of spaces needed)
Second line: A string of up to 256 characters representing the street, where `_` (underscore) represents an empty parking space and any other character represents an occupied space.

Output

Output a single line containing the street string where the first sequence of consecutive `_` characters (of length equal to car size) is replaced with `*` characters if the car can park. The original string is printed unchanged if no parking spot is available

Example 1

Input

3

__X__X__

Output

__X__X__

Example 2

Input

4

X_X_X___X____X

Output

X_X_X___X****X

Python

```python
def find_parking(car_size, street):
    """
    Find a parking spot for a car in a street.

    Args:
        car_size: Integer representing the size of the car
        street: String up to 256 chars representing the street
                Empty spots are '_', occupied spots are other characters

    Returns:
        String with '_' replaced by '*' where the car can fit,
        or the original string if no parking spot is available
    """
    # If car size is 0 or negative, return original street
    if car_size <= 0:
        return street

    # If street is shorter than car size, can't park
    if len(street) < car_size:
        return street

    # Search for a continuous sequence of '_' that fits the car
    for i in range(len(street) - car_size + 1):
        # Check if we have enough consecutive empty spots
        if all(street[i + j] == '_' for j in range(car_size)):
            # Found a parking spot! Replace with '*'
            result = street[:i] + '*' * car_size + street[i + car_size:]
            return result

    # No parking spot found, return original street
    return street
```

```
# Example usage and test cases
if __name__ == "__main__":
    """
    # Test case 1: Car fits in the middle
    print(find_parking(3, "__X__X__"))  # Expected: "***X__X__"

    # Test case 2: Car fits at the end
    print(find_parking(2, "X_X___"))    # Expected: "X_X**_"

    # Test case 3: Car doesn't fit anywhere
    print(find_parking(3, "X_X_X_"))    # Expected: "X_X_X_" (no change)

    # Test case 4: Exact fit
    print(find_parking(4, "____"))      # Expected: "****"

    # Test case 5: No empty spots
    print(find_parking(2, "XXXX"))      # Expected: "XXXX" (no change)

    # Interactive mode
    print("\n--- Interactive Mode ---")
    """

    size = int(input())
    street = input()[:256]  # Limit to 256 chars
    result = find_parking(size, street)
    print(result)
```

10 Code Warrior

5 points

Introduction

You are a brave warrior exploring a dungeon! Your journey can be represented as a string where each character represents a square in the dungeon:

- _ represents an empty square (you can walk through it)
- G represents a Goblin
- S represents a Giant Spider
- E represents a Skeleton
- T represents a Troll

Your task is to write a program that outputs the sequence of actions the warrior must take to traverse the entire dungeon. The actions are:

- w (walk) when the square is empty
- f (fight) when the square is occupied by a creature

To defeat creatures, you must fight them multiple times:

- Goblins (G): 1 fight
- Skeletons (E): 2 fights
- Giant Spiders (S): 3 fights
- Trolls (T): 4 fights

Once a creature is defeated, the square becomes empty and the warrior can walk over it (one additional w action).

Remember

After defeating a creature, you need to add one more action (w) to walk through the now-empty square!

Input

A single line containing a string representing the dungeon. The string will only contain the characters _, G, S, E, and T.

Output

A single line containing a string of actions (w for walk, f for fight) that the warrior must perform to traverse the entire dungeon.

Example 1

Input

```
___G_
```

Output

```
wwwfww
```

Example 2

Input

```
_E_S
```

Output

```
wffwwfffw
```

Example 3

Input

```
GT_
```

Output

```
fwffffww
```

Python

```python
def code_warrior(dungeon):
    """
    Given a dungeon string, return the sequence of actions needed to traverse
it.

    Args:
        dungeon: String containing _, G, S, E, T representing the dungeon

    Returns:
        String of actions (w for walk, f for fight)
    """
    # Map creatures to number of fights needed
    creature_fights = {
        'G': 1,  # Goblin
        'E': 2,  # Skeleton
        'S': 3,  # Giant Spider
        'T': 4   # Troll
    }

    actions = ""

    for square in dungeon:
        if square == '_':
            # Empty square - just walk
            actions += 'w'
        else:
            # Creature - fight the required number of times, then walk through
            fights_needed = creature_fights[square]
            actions += 'f' * fights_needed
            actions += 'w'

    return actions


if __name__ == "__main__":
    # Read input from command line
    dungeon = input()
    result = code_warrior(dungeon)
    print(result)
```

11 Count Peaks

7 points

Introduction

You're training to become a mountain guide, and your instructor has given you a topographic map... well, kind of. Instead of a real map, it's just a series of elevation numbers. "Count the peaks!" they shout enthusiastically. "A peak is any point that's higher than both its neighbors!"

You stare at the numbers: 1 3 2 5 4 8 6. The instructor points: "See? The 3 is higher than 1 and 2, so it's a peak! The 5 is higher than 2 and 4, another peak! And that 8? Higher than 4 and 6, definitely a peak! That's 3 peaks total!"

But what about the numbers at the edges? "Those can't be peaks," explains your instructor. "You need neighbors on BOTH sides to be a peak. The first and last numbers only have one neighbor, so they don't count."

Now you need to write a program to count peaks automatically, because counting by hand is getting tiring and you have a LOT of maps to analyze!

Peak Definition:
- A value is a peak if it's strictly greater than BOTH its neighbors
- The first and last elements cannot be peaks (they don't have two neighbors)
- For a series with fewer than 3 elements, there are no peaks

Input

A line containing `n` space-separated integers representing the elevations. Each elevation is between -1,000,000 and 1,000,000.

Output

Output a single integer: the number of peaks in the series. If number of peaks is zero then print “There are no peaks!”.

Example 1

Input

```
1 3 2 5 4 8 6
```

Output

```
3
```

Example 2

Input

```
1 2 3 4 5
```

Output

```
There are no peaks!
```

Python

```python
elevations = list(map(int, input().split()))
n = len(elevations)

# Count peaks
peaks = 0

# Check each element except first and last
for i in range(1, n - 1):
    if elevations[i] > elevations[i - 1] and elevations[i] > elevations[i + 1]:
        peaks += 1

if peaks == 0:
    print("There are no peaks!")
else:
    print(peaks)
```

12 Garden Planner

7 points

Introduction

Maria is designing her dream garden for the spring season and needs to calculate how much grass seed to buy. Her garden has several sections with different shapes - some are rectangular flower beds and others are circular ponds. To determine the exact amount of grass seed needed, she must calculate the total area to be covered.

Each bag of grass seed covers exactly 1 square meter, so Maria needs precise area calculations for her budget. Help Maria by computing the area of each garden section!

Hint: Don't hardcode π constant! Use your language's built-in constant instead (**math.pi** in Python, **M_PI** in C/C++).

Input

Input begins with a number n, followed by n shape descriptions.

For a rectangular section, the line contains the word "rectangle" followed by two strictly positive real numbers: its length and its width.

For a circular section, the line contains the word "circle" followed by a strictly positive real number: its radius.

Output

For each shape description, print its area in square meters with 6 digits after the decimal point.

Example 1

Input

```
2
rectangle 5.0 3.0
circle 2.5
```

Output

```
15.000000
19.634954
```

Example 2

Input

```
4
circle 1.0
rectangle 2.5 4.0
circle 3.2
rectangle 7.5 2.8
```

Output

```
3.141593
10.000000
32.169909
21.000000
```

Example 3

Input

```
1
rectangle 10.5 8.3
```

Output

```
87.150000
```

Python

```python
import math

def calculate_area(shape_type, *dimensions):
    """
    Calculate area based on shape type.

    :param shape_type: 'rectangle' or 'circle'
    :param dimensions: length and width for rectangle, radius for circle
    :return: Area of the shape
    """
    if shape_type == "rectangle":
        length, width = dimensions
        return length * width
    elif shape_type == "circle":
        radius = dimensions[0]
        return math.pi * radius * radius
    else:
        return 0.0

def main():
    n = int(input().strip())

    for _ in range(n):
        line = input().strip().split()
        shape_type = line[0]

        if shape_type == "rectangle":
            length = float(line[1])
            width = float(line[2])
            area = calculate_area(shape_type, length, width)
        elif shape_type == "circle":
            radius = float(line[1])
            area = calculate_area(shape_type, radius)
        else:
            area = 0.0

        print(f"{area:.6f}")

if __name__ == "__main__":
    main()
```

13 The Time Rebels Of France!

7 points

Introduction

A long, long time ago (okay, about 230 years ago), some very serious French revolutionaries had a very serious idea: "*What if we made time more... decimal?*"

So in 1793, the French National Convention decided that a day would no longer have 24 hours, but 10 hours! They said: *"Voilà! One day, ten hours. Each hour has 100 minutes, and each minute has 100 seconds. Easy-peasy!"*

Of course, not everyone loved it. Bakers were confused, roosters didn't know when to crow, and people missed lunch because their watches were weird. Still, this Decimal Time was officially used for a while – until everyone realized that the Earth didn't care about their math.
So now, it's your job to help translate between Metric (normal) time and Decimal time! Because when the revolutionaries need their croissants on time, every second counts!

Input

It consists of two lines, the first line with a single word telling what type of time you are given:

- "Metric" → means normal time (like we use today)
- "Decimal" → means French Revolutionary time (10-hour days)

The second line the time in this format: HH:MM:SS

Output

You must print the converted time in the same HH:MM:SS format, rounding seconds to the nearest whole number.

Example 1

Input

Metric

14:30:00

Output

06:04:17

Example 2

Input

Decimal

05:00:00

Output

12:00:00

Python

```python
# Decimal ⇔ Metric Time Converter (Correct rounding)

time_type = input().strip()
time_str = input().strip()

h, m, s = map(int, time_str.split(':'))

if time_type == "Metric":
    total_metric_seconds = h * 3600 + m * 60 + s
    # Multiply first, then divide using rounding
    total_decimal_seconds = round((total_metric_seconds * 100000) / 86400)

    decimal_h = total_decimal_seconds // 10000
    decimal_m = (total_decimal_seconds % 10000) // 100
    decimal_s = total_decimal_seconds % 100
    print(f"{decimal_h:02d}:{decimal_m:02d}:{decimal_s:02d}")

elif time_type == "Decimal":
    total_decimal_seconds = h * 10000 + m * 100 + s
    total_metric_seconds = round((total_decimal_seconds * 86400) / 100000)

    metric_h = total_metric_seconds // 3600
    metric_m = (total_metric_seconds % 3600) // 60
    metric_s = total_metric_seconds % 60
    print(f"{metric_h:02d}:{metric_m:02d}:{metric_s:02d}")

else:
    print("Invalid input type. Must be 'Metric' or 'Decimal'.")
```

14 Spam Filter

7 points

Introduction

You are developing an anti-spam system for an e-mail service and need to implement a filter that detects promotional messages. Your system should flag emails that contain variations of the word 'sale' used by spammers to bypass filters, both in upper and lower case or with number substitutions: `sale`, `sa1e`, `s4le`, `sal3`, `5ale`, `5a1e`, `s4l3` and `5al3`.

Remember that also `SALE`, `Sa1E`, `S4Le`... are spam! Don't let them through!

Input

A single line containing the subject line of an email.

Output

The filter should respond 'Email flagged as spam.' or 'Email appears legitimate.' depending on whether they contain the forbidden promotional words described above.

Example 1

Input

```
Amazing sale on electronics today only!
```

Output

```
Email flagged as spam.
```

Example 2

Input

```
Weekly newsletter with company updates
```

Output

```
Email appears legitimate.
```

Example 3

Input

```
Don't miss our 5a1e on winter clothes!
```

Output

```
Email flagged as spam.
```

Python

```python
# solution.py

def is_spam(subject_line):
    """
    Determines if an email subject line is spam based on variations of the word 'sale'.

    :param subject_line: String representing the email subject line.
    :return: True if spam, False if legitimate.
    """
    # Convert to lowercase for case-insensitive matching
    subject_lower = subject_line.lower()

    # Define all variations of 'sale' that indicate spam
    spam_variations = [
        'sale', 'sa1e', 's4le', 'sal3',
        '5ale', '5a1e', 's413', '5al3'
    ]

    # Check if any spam variation is present in the subject line
    for variation in spam_variations:
        if variation in subject_lower:
            return True

    return False

if __name__ == "__main__":
    subject_line = input().strip()

    if is_spam(subject_line):
        print("Email flagged as spam.")
    else:
        print("Email appears legitimate.")
```

15 Palindrome Maker

9 points

Introduction

You're texting your friend and accidentally type "racecar" - they reply "Nice palindrome!" You type "hello" and they say "Not a palindrome, but you could make it 'hellolleh'!"

You're fascinated. A palindrome is a word that reads the same forwards and backwards, like "racecar", "level", or "mom". But what's really interesting is: how many characters do you need to ADD to any word to turn it into a palindrome? For example:

- "race" → add "car" to make "racecar" (3 characters added)
- "abc" → add "ba" to make "abcba" (2 characters added)
- "mom" → already a palindrome! (0 characters added)

Your task is to find the MINIMUM number of characters you need to add to make any string a palindrome. You can only add characters at the end of the string!

Think about it: if "race" + "car" = "racecar", those 3 characters "car" are just the reverse of the first 3 characters "rac"!

Input

A single line containing a string s (1 ≤ length ≤ 1000) consisting only of lowercase letters.

Output

Output a single integer: the minimum number of characters to add to the end of the string to make it a palindrome. Return 0 if the word is already a palindrome.

Example1

Input

```
race
```

Output

```
3
```

Example 2

Input

```
mom
```

Output

```
0
```

Python

```python
#!/usr/bin/env python3

def is_palindrome(s):
    """Check if a string is a palindrome."""
    return s == s[::-1]

s = input().strip()

# Check if we need to add 0, 1, 2, ... characters
for chars_to_add in range(len(s)):
    # Build the candidate palindrome by adding first chars_to_add characters
reversed
    candidate = s + s[:chars_to_add][::-1]

    if is_palindrome(candidate):
        print(chars_to_add)
        break
```

16 From One Chord To Another

9 points

Introduction

Backstage at the spring talent show, a small group of 16–18-year-olds huddle over smudged lyric sheets. Their goal: adapt a favorite song so every vocalist can sing it comfortably, no matter their range. Your task is to help them transpose each chord by a fixed number of half tones, letting the arrangement travel from one key to another without losing its pulse.

Hint: The chromatic ladder is circular: C, C#, D, D#, E, F, F#, G, G#, A, A#, B, then back to C. Each step is exactly one half tone.

Input

The first line contains an integer n ($1 \le n \le 200$) - the number of chords in the song.
The second line contains an integer k ($0 \le k \le 1{,}000$) - how many half tones to transpose (only non-negative shifts are allowed).
The next n lines each contain a chord symbol from the set {C, C#, D, D#, E, F, F#, G, G#, A, A#, B}.

Output

Print n lines. For each original chord, output the chord that lies k half tones away on the circular chromatic scale.

Example 1

Input

```
3
2
C
D
F#
```

Output

```
D
E
G#
```

Example 2

Input

```
4
0
C
F
A#
G
```

Output

```
C
F
A#
G
```

Example 3

Input

```
4
12
C#
E
G
A#
```

Output

```
C#
E
G
A#
```

Example 4

Input

```
6
26
C
D#
F
G
A
B
```

Output

```
D
F
G
A
B
C#
```

C++

```cpp
#include <cstdio>
#include <cstring>

int main() {
    static const char *NOTES[12] = {
        "C", "C#", "D", "D#", "E", "F",
        "F#", "G", "G#", "A", "A#", "B"
    };

    int n, k;
    if (std::scanf("%d", &n) != 1) {
        return 0;
    }
    if (std::scanf("%d", &k) != 1) {
        return 0;
    }

    char chord[16];
    while (n--) {
        if (std::scanf("%15s", chord) != 1) {
            return 0;
        }

        int index = -1;
        for (int i = 0; i < 12; ++i) {
            if (std::strcmp(chord, NOTES[i]) == 0) {
                index = i;
                break;
            }
        }

        if (index == -1) {
            std::puts("?");
        } else {
            int shifted = index + k;
            shifted %= 12;
            if (shifted < 0) {
                shifted += 12;
            }
            std::puts(NOTES[shifted]);
        }
    }
    return 0;
}
```

17 Alphabet Rhomboid

10 points

Introduction

The royal artisan of Alphabeta needs help designing rhomboid-shaped letter banners. Given two lowercase letters, print a symmetric rhomboid pattern starting from the first letter, expanding one character per line until reaching the second letter, then shrinking back to the first. Can you help the artisan by writing this program?

Input

Two lowercase letters, each on a separate line. The first letter represents the starting character, and the second letter represents the ending character. It is guaranteed that the second letter comes after the first letter alphabetically.

Output

Print an alphabet rhomboid pattern starting from the first letter, incrementing by one character each line until reaching the second letter, then decrementing back to the first letter. Each line contains repeated occurrences of the corresponding letter, forming a symmetric rhomboid shape.

Example 1

Input

```
a
b
```

Output

```
 a
bb
a
```

Example 2

Input

```
c
e
```

Output

```
  c
 dd
eee
dd
c
```

Example 3

Input

```
t
z
```

Output

```
      t
     uu
    vvv
   wwww
  xxxxx
 yyyyyy
zzzzzzz
yyyyyy
xxxxx
wwww
vvv
uu
t
```

Python

```python
begin = input()
end = input()

length = ord(end) - ord(begin)

# First half of the romboid
next = begin
for i in range(0, length+1):
    spaces = ' ' * (length-i)
    print(spaces + (i+1)*next)
    next = chr(ord(next) + 1)

# Second half of the romboid
next = chr(ord(end) - 1)
for i in range(length -1 , -1, -1):
    print((i+1)*next)
    next = chr(ord(next) - 1)
```

18 Laser Gate Alignment

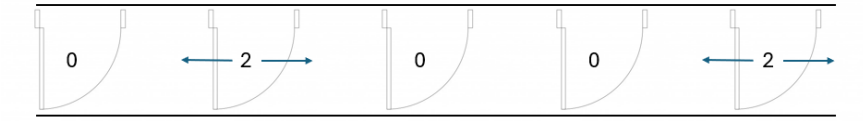
11 points

Introduction

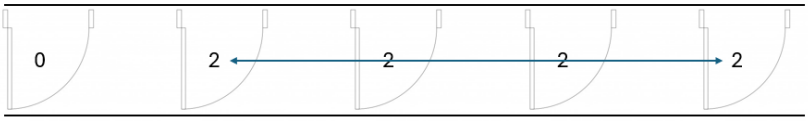
In a secret science lab, there is a long corridor filled with gates of two kinds: active and inactive. Active gates have laser emitters on both sides, ready to fire their beams. Inactive gates have no emitters and allow laser beams to pass through. When two lasers of the same frequency connect, all inactive gates between them light up!

The gates are represented by numbers:

- A positive number represents an active gate with laser emitters, where the value indicates its frequency.
- A 0 represents an inactive gate.



If two emitters of the same frequency face each other, and only inactive gates (0) lie between them, their beams connect and activate all the gates in between. However, if any other emitter (with a different frequency) is present in between, the beams do not synchronize, and no gates light up.



Your challenge is to write a program that figures out which gates will light up in the lab corridor.

Input

A single line with positive integers and zeros representing the laser and gates of the lab corridor.

Output

A single line with the final laser beam configuration of the corridor.

Example 1

Input

```
0 2 0 0 2
```

Output

```
0 2 2 2 2
```

Example 2

Input

```
0 2 0 1 0 2
```

Output

```
0 2 0 1 0 2
```

Example 3

Input

```
0 2 0 2 5 0 0 5 1
```

Output

```
0 2 2 2 5 5 5 5 1
```

Example 4

Input

```
3 0 0 3 0 3
```

Output

```
3 3 3 3 3 3
```

Python

```python
def solution(lasers):
    elements = lasers.split()
    n = len(elements)

    # Convert to list for easier manipulation
    result = elements[:]

    # For each position, check if it's an emitter
    for i in range(n):
        if elements[i] != '0':
            frequency = elements[i]

            # Look for matching emitter to the right
            for j in range(i + 1, n):
                if elements[j] == frequency:
                    # Check if path between i and j contains only zeros
                    blocked = False
                    for k in range(i + 1, j):
                        if elements[k] != '0':
                            blocked = True
                            break

                    # If not blocked, activate all gates between emitters with
the frequency
                    if not blocked:
                        for k in range(i + 1, j):
                            result[k] = frequency  # Use frequency instead of
'1'
                        break
                elif elements[j] != '0':
                    # Different emitter blocks the beam
                    break

    return ' '.join(result)

# The input string is read from standard input as a list [3, 0, 0, 3, 0, 3]
lasers = input()

result = solution(lasers)
print(result)
```

19 Oil Well Exploration

12 points

Introduction

You are a geologist tasked with analyzing seismic data to identify potential oil well locations. The data is represented as a rectangular grid of numbers, where each cell contains an integer value. These integers represent the "seismic activity" of that specific location, which can indicate the presence of oil deposits. However, not all values are positive—some locations may have geological formations that are not conducive to oil deposits, represented by negative numbers. The range of these values is between -1000 and 1000.

Your goal is to help your team evaluate specific regions of the seismic data to determine their total seismic activity. This will help identify areas that are most likely to contain oil deposits. To do this, you are given a list of subregions (rectangular areas) on the map that need to be analyzed. For each subregion, you need to calculate the sum of all seismic activity values within it.

Input

The first line contains three integers: the width (w) and the height (h) of the seismic map, plus the number of subregions (q) to analyze.

The next h lines describe the seismic map itself. Each line contains w integers separated by spaces, representing the seismic activity for that row.

After the seismic map data, there are q lines describing the subregions to analyze. Each line contains four integers: x1, y1, x2, and y2 having always $1 \le x1 \le x2 \le w$ and $1 \le y1 \le y2 \le h$. These represent two opposing corners of a rectangular subregion on the map. The coordinates are indexed starting from 1 (not 0) from the upper left corner of the seismic map. The coordinates include their respective edges in the subregion.

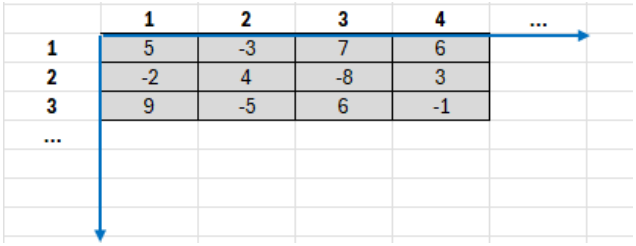
Output

For each subregion, print a single integer on its own line: the sum of all seismic activity values within that subregion.

Example

Input

```
4 3 2
5 -3 7 6
-2 4 -8 3
9 -5 6 -1
1 1 3 2
2 2 4 3
```



Output

```
3
-1
```

Python

```python
def analyze_seismic_data():
    # Read the first line: width, height, and number of queries
    w, h, q = map(int, input().split())

    # Read the seismic map
    seismic_map = []
    for iter in range(h):
        row = list(map(int, input().split()))
        seismic_map.append(row)

    # Process each subregion query
    for iter in range(q):
        x1, y1, x2, y2 = map(int, input().split())

        # Normalize coordinates (handle reverse order)
        x_min, x_max = min(x1, x2), max(x1, x2)
        y_min, y_max = min(y1, y2), max(y1, y2)

        # Calculate the sum for the subregion
        # Note: coordinates are 1-indexed, so we need to adjust
        total = 0
        for row in range(y_min - 1, y_max):  # y_min-1 to y_max-1 inclusive
            for col in range(x_min - 1, x_max):  # x_min-1 to x_max-1 inclusive
                total += seismic_map[row][col]

        print(total)

if __name__ == "__main__":
    analyze_seismic_data()
```

20 Find The Unique Pair

13 points

Introduction

You're a cryptographer working for a secret intelligence agency, and you've just intercepted an encrypted message from an enemy organization. After weeks of analysis, your team discovered that the encryption key is based on a peculiar mathematical sequence with a strict uniqueness property.

The sequence of numbers follows a simple but strict rule:
- Start with 1 and 2
- Each subsequent number must be the unique sum of two distinct earlier numbers in the sequence
- Unique means that number can be expressed as the sum of two earlier numbers in exactly one way

Let's see how it works:
- Start: 1, 2
- 3 = 1+2 (only one way) √ → 1, 2, 3
- 4 = 1+3 (only one way, since 2+2 doesn't count - numbers must be distinct) √ → 1, 2, 3, 4
- 5 = 1+4 OR 2+3 (two ways!) ✗ Skip 5
- 6 = 2+4 (only one way, since 1+5 doesn't work as 5 isn't in sequence) √ → 1, 2, 3, 4, 6
- 7 = 1+6 OR 3+4 (two ways!) ✗ Skip 7
- 8 = 2+6 (only one way) √ → 1, 2, 3, 4, 6, 8
- 11 = 3+8 (only one way) √ → 1, 2, 3, 4, 6, 8, 11

The sequence continues: 1, 2, 3, 4, 6, 8, 11, 13, 16, 18, 26, 28, 36, 38, 47...

Your mission: You've been given a number from an intercepted message. You need to determine if it's in this special sequence, and if it is, reveal the unique pair of numbers that sum to it. This pair is the decryption key!

Input

A single positive integer M ($1 \le M \le 500$)

Output

- If M is not in the sequence, print: M is not in sequence.
- If M is in the sequence:
    - If M is 1 or 2 (starting numbers), print: M is in sequence (starter)
    - Otherwise, print: M is in sequence: A + B where A and B are the unique pair (A < B)

Example 1

Input

```
11
```

Output

```
11 is in sequence: 3 + 8
```

Example 2

Input

1

Output

```
1 is in sequence (starter)
```

Example 3

Input

5

Output

```
5 is not in sequence
```

Python

```python
def generate_ulam_with_pairs(limit):
    """Generate Ulam numbers up to limit, storing the pair that created each."""
    ulam = [1, 2]
    pairs = {1: None, 2: None}  # Starters have no pair

    candidate = 3
    while candidate <= limit:
        # Count how many ways we can form this candidate
        count = 0
        found_pair = None

        for i in range(len(ulam)):
            for j in range(i + 1, len(ulam)):
                if ulam[i] + ulam[j] == candidate:
```

```
                    count += 1
                    if count == 1:
                        found_pair = (ulam[i], ulam[j])
                    if count > 1:
                        break
            if count > 1:
                break

        # If exactly one way to form it, add to sequence
        if count == 1:
            ulam.append(candidate)
            pairs[candidate] = found_pair

        candidate += 1

    return ulam, pairs

# Main program

# Read input
m = int(input())

# Generate Ulam numbers up to m (or a bit beyond to be safe)
ulam_numbers, pairs = generate_ulam_with_pairs(m)

# Check if m is an Ulam number
if m in pairs:
    if m == 1 or m == 2:
        print(f"{m} is in sequence (starter)")
    else:
        a, b = pairs[m]
        print(f"{m} is in sequence: {a} + {b}")
else:
    print(f"{m} is not in sequence")
```

21 Ghost In The Static

13 points

Introduction

Chiba City, the Sprawl. You're a console cowboy jacked into the Matrix, navigating the neon-lit architecture of cyberspace. Wintermute, the rogue AI, has been broadcasting fragmented coordinates to a hidden data vault — but the signal is corrupted, full of recursive loops and static noise. To pinpoint the vault's location, you need to isolate the longest stable sequence — a clean run of data pulses where no signal repeats. Duplicates mean interference, interference means losing the trail forever.

Given a string, find the longest substring without duplicate characters. The program should identify and return the first occurrence of the longest substring that contains only unique characters (no character appears more than once in the substring).

Input

The program accepts a single line of input containing a string. The string can contain:

- Letters (uppercase and lowercase)
- Numbers
- Special characters
- Spaces

The maximum length of the input string is 256 characters.

Output

The program outputs a single line containing the first longest substring without duplicate characters. If there are multiple substrings of the same maximum length, the program returns the first one found (leftmost in the original string).

Example 1

Input

abcabcbb

Output

abc

Example 2

Input

pwwdasfasdfasdfskfdfsafew111223aed

Output

safew1

Python

```python
def longest_substring_without_duplicates(s):
    """
    Find the length and the actual longest substring without duplicate
characters.

    Args:
        s: Input string

    Returns:
        tuple: (length, substring)
    """
    if not s:
        return 0, ""

    char_index = {}
    max_length = 0
    max_start = 0
    start = 0

    for i, char in enumerate(s):
        # If character is already in current window, move start pointer
        if char in char_index and char_index[char] >= start:
            start = char_index[char] + 1

        # Update character's latest position
        char_index[char] = i

        # Update max length and starting position if current window is longer
        current_length = i - start + 1
        if current_length > max_length:
            max_length = current_length
```

```
            max_start = start

    longest_substr = s[max_start:max_start + max_length]
    return max_length, longest_substr


def main():
    """
    # Example 1
    print("\n--- Example 1 ---")
    test1 = "abcabcbb"
    length1, substr1 = longest_substring_without_duplicates(test1)
    print(f"Input: '{test1}'")
    print(f"Length: {length1}")
    print(f"First longest substring: '{substr1}'")

    # Example 2
    print("\n--- Example 2 ---")
    test2 = "pwwdasfasdfasdfskfdfsafew111223aed"
    length2, substr2 = longest_substring_without_duplicates(test2)
    print(f"Input: '{test2}'")
    print(f"Length: {length2}")
    print(f"First longest substring: '{substr2}'")
    """
    # Interactive mode
    user_input = input().strip()

    length, substr = longest_substring_without_duplicates(user_input)
    print(substr)


if __name__ == "__main__":
    main()
```

22 Bullseye

13 points

Introduction

🎯 It's game night at your friend's house and someone brought a dartboard! Everyone's having a blast throwing darts, but there's chaos when it comes to scoring. "I hit near the center!" claims one friend. "That's gotta be like... 80 points?" Another friend disagrees: "No way, you were way off! That's maybe 20 points!"

You realize nobody actually knows how dartboard scoring works, and people are just making up numbers. Time to write a program that calculates the REAL score!

In this simplified dartboard scoring system, the board is divided into concentric circles:

- Bullseye (center): If the dart lands exactly at the center (0, 0) or within distance 1 from center → 100 points
- Inner ring: Distance greater than 1 but ≤ 5 from center → 50 points
- Middle ring: Distance greater than 5 but ≤ 10 from center → 25 points
- Outer ring: Distance greater than 10 but ≤ 20 from center → 10 points
- Off the board: Distance greater than 20 from center → 0 points

The dartboard's center is at coordinates (0, 0), and you'll be given where each dart landed as (x, y) coordinates.

Distance formula: The distance from center is calculated as: $d = \sqrt{x^2 + y^2}$

Input

The first line contains an integer n ($1 \le n \le 100$), the number of darts thrown.
The next n lines each contain two space-separated floating-point numbers x and y ($-30.0 \le x, y \le 30.0$), representing where each dart landed.

Output

Output a single integer: the total score from all darts.

Example 1

Input

```
3
0.0 0.0
3.0 4.0
15.0 0.0
```

Output

```
160
```

Example 2

Input

```
4
0.5 0.5
10.0 10.0
20.0 20.0
-25.0 5.0
```

Output

```
110
```

Python

```python
#!/usr/bin/env python3
import math

n = int(input())
total_score = 0

for _ in range(n):
    x, y = map(float, input().split())

    # Calculate distance from center
    distance = math.sqrt(x * x + y * y)

    # Determine score based on distance
    if distance <= 1.0:
        score = 100
    elif distance <= 5.0:
        score = 50
    elif distance <= 10.0:
        score = 25
    elif distance <= 20.0:
        score = 10
    else:
        score = 0

    total_score += score

print(total_score)
```

23 Molecular Mass Puzzle

14 points

Introduction

In chemistry, scientists often identify an unknown molecule by measuring its exact mass using a technique called mass spectrometry. Each element has a well-defined isotopic mass that has been determined with very high precision (many decimals), and the exact mass of a molecule is exactly the sum of the isotopic masses of the atoms that compose it.

Element	Isotopic Mass (u)
H (Hydrogen)	1.00782503223
C (Carbon)	12.00000000000
O (Oxygen)	15.99491461957

Your task is to determine the molecular formula, the counts of C, H and O, such that the sum of their isotopic masses equals the given mass exactly.
You may safely assume:

- A valid combination of C, H, and O always exists. No other atoms are present.
- The masses match exactly to the precision given.

HINTS:
Molecules won't have more than 20 of any single atom type.
When precision is needed in adding decimal numbers, computers accumulate something called floating-point rounding errors. To handle these errors, you would need to compare with an epsilon tolerance (e.g., abs(total - target) < 1e-12) due to floating-point rounding errors.

Input

A single decimal number representing the exact mass of a mystery molecule.

Output

A string containing the molecular formula: C<cnt>H<cnt>O<cnt>, where cnt represents the number of atoms. If an atom isn't present in the compound, its count is 0.

Example 1

Input

```
18.01056468403
```

Output

```
C0H2O1
```

Example 2

Input

```
16.03130012892
```

Output

```
C1H4O0
```

Example 3

Input

```
46.04186481295
```

Output

```
C2H6O1
```

Example 4

Input

```
60.02112936806
```

Output

```
C2H4O2
```

Example 5

Input

```
28.03130012892
```

Output

```
C2H4O0
```

Example 6

Input

```
44.02621474849
```

Output

```
C2H4O1
```

Example 7

Input

```
44.06260025784
```

Output

```
C3H8O0
```

Example 8

Input

```
88.01604398763
```

Output

```
C3H4O3
```

Python

```python
"""
Molecular Mass Puzzle

This solution finds which combination of Carbon (C), Hydrogen (H), and Oxygen
(O)
atoms creates a molecule with the exact mass given.

Key Concept:
We try different combinations of atoms until we find one where the total mass
matches exactly. It's like solving a puzzle by trying different pieces!

Strategy:
1. Start with 0 atoms of each type
2. Try adding atoms one by one
3. Check if the total mass matches
4. When we find a match, we're done!
"""

# We need the Decimal module for precise calculations with many decimal places
```

```python
# Regular floating-point numbers (like 1.5 or 3.14) can have tiny errors
# Decimal numbers are exact, which is critical for chemistry!
from decimal import Decimal

# These are the exact masses of each atom type (in atomic mass units)
# Scientists have measured these values very precisely
MASS_H = Decimal('1.00782503223')   # Hydrogen atom mass
MASS_C = Decimal('12.00000000000')  # Carbon atom mass
MASS_O = Decimal('15.99491461957')  # Oxygen atom mass


def calculate_total_mass(num_carbon, num_hydrogen, num_oxygen):
    """
    Calculate the total mass of a molecule given the number of each atom type.

    Parameters:
    - num_carbon: How many Carbon atoms (C)
    - num_hydrogen: How many Hydrogen atoms (H)
    - num_oxygen: How many Oxygen atoms (O)

    Returns:
    - The total mass as a Decimal number

    Example: Water (H2O) has 0 Carbon, 2 Hydrogen, 1 Oxygen
             Total mass = 0*12.0 + 2*1.008 + 1*15.995 = 18.011
    """
    # Multiply the number of each atom by its mass, then add them all up
    total = (num_carbon * MASS_C) + (num_hydrogen * MASS_H) + (num_oxygen *
MASS_O)
    return total


def solution(target_mass):
    """
    Find the molecular formula (counts of C, H, O) that matches the target mass.

    Parameters:
    - target_mass: The exact mass we're trying to match (as a Decimal)

    Returns:
    - A string like "C2H4O1" showing the number of each atom

    How it works:
    We use three nested loops to try every possible combination:
    - The outer loop tries different numbers of Oxygen atoms (0, 1, 2, ...)
    - The middle loop tries different numbers of Carbon atoms (0, 1, 2, ...)
    - The inner loop tries different numbers of Hydrogen atoms (0, 1, 2, ...)
```

```
    For each combination, we check if the total mass matches our target.
    """

    # Convert the input to Decimal if it isn't already
    # This ensures we have the precision we need
    target_mass = Decimal(str(target_mass))

    # We need to set limits on how many atoms to try
    # Most molecules won't have more than 20 of any single atom type
    # We start with Oxygen because it's the heaviest atom
    # This makes our search more efficient
    max_atoms = 20

    # Try different numbers of Oxygen atoms (O is heaviest, so we try it first)
    for num_oxygen in range(max_atoms):

        # Calculate how much mass the oxygen atoms contribute
        mass_from_oxygen = num_oxygen * MASS_O

        # If just the oxygen is already heavier than our target, skip to next
        # (No point trying to add more atoms if we're already too heavy!)
        if mass_from_oxygen > target_mass:
            break  # Stop trying more oxygen atoms

        # Calculate how much mass we still need after accounting for oxygen
        remaining_mass_after_oxygen = target_mass - mass_from_oxygen

        # Now try different numbers of Carbon atoms
        for num_carbon in range(max_atoms):

            # Calculate how much mass the carbon atoms contribute
            mass_from_carbon = num_carbon * MASS_C

            # If oxygen + carbon is already too heavy, try less carbon
            if mass_from_carbon > remaining_mass_after_oxygen:
                break  # Stop trying more carbon atoms

            # Calculate how much mass we still need after oxygen and carbon
            remaining_mass_after_carbon = remaining_mass_after_oxygen -
mass_from_carbon

            # Now try different numbers of Hydrogen atoms
            for num_hydrogen in range(max_atoms):

                # Calculate the total mass with this combination
```

```
                total_mass = calculate_total_mass(num_carbon, num_hydrogen,
num_oxygen)

                # Check if we found an exact match!
                # We compare Decimal numbers directly - they handle precision
correctly
                if total_mass == target_mass:
                    # Success! We found the right combination
                    # Format it as "C#H#O#" and return
                    result = f"C{num_carbon}H{num_hydrogen}O{num_oxygen}"
                    return result

                # If we've added too much mass with hydrogen, stop trying more H
                # Hydrogen is the lightest, so if H makes it too heavy, we need
                # to try a different combination of C and O
                if total_mass > target_mass:
                    break  # Stop trying more hydrogen atoms

    # If we get here, we didn't find a match
    # According to the problem, this shouldn't happen!
    return "No solution found"

# ============================================================================
# MAIN PROGRAM
# ============================================================================

massRead = input()
mass1 = Decimal(massRead)
result1 = solution(mass1)
print(result1)
```

24 Pizza Delivery Disaster

15 points

Introduction

The night before CodeWars, the organizing staff is working late setting up the contest venue. As tradition, they order pizzas from Tony's Pizza Palace to fuel their all-night preparation.

Tony's pizza boxes have special magnetic locks that only open when the delivery address is divisible by 13 (Tony's lucky number). But disaster strikes! Tony's nephew Joey mixed up all the address number stickers, and now they're scattered on the floor!

Joey has n different digit stickers (digits 1-9) and must figure out which house numbers he can form that will open the pizza boxes. The hungry CodeWars staff needs their pizza to finish setting up the contest!

Help Joey find all possible house numbers (multiples of 13) using the available digits, so the staff can eat and complete the setup.

Input

The input consists of two lines. First line has a number $1 \leq n \leq 9$, followed by a second line with n different digits between 1 and 9.

Output

Print all house numbers (multiples of 13) that can be formed using the available digit stickers, sorted in ascending order. If no valid house numbers can be formed, print "No pizza for anyone!" (without quotes).

Example 1

Input

```
3
2 6 5
```

Output

```
26
52
65
```

Example 2

Input

```
2
1 3
```

Output

```
13
```

Example 3

Input

```
4
1 3 5 7
```

Output

```
13
351
715
1573
5317
```

Example 4

Input

```
2
2 3
```

Output

```
No pizza for anyone!
```

Python

```python
def generate_permutations(elements, length):
    """
    Generate all permutations of given length from elements.

    :param elements: List of elements to permute
    :param length: Length of each permutation
    :return: List of all permutations
    """
    if length == 0:
        return [[]]

    if length == 1:
        return [[elem] for elem in elements]

    result = []
    for i, elem in enumerate(elements):
        # Get remaining elements (excluding current)
        remaining = elements[:i] + elements[i+1:]
        # Recursively generate permutations of remaining elements
        for perm in generate_permutations(remaining, length - 1):
            result.append([elem] + perm)

    return result

def find_pizza_houses(digits):
```

```
    """
    Find all house numbers (multiples of 13) that can be formed from given
digits.

    :param digits: List of available digit stickers
    :return: Sorted list of valid house numbers
    """
    valid_houses = set()

    # Try all possible permutations of all possible lengths
    for length in range(1, len(digits) + 1):
        perms = generate_permutations(digits, length)
        for perm in perms:
            # Convert permutation to number
            number = int(''.join(map(str, perm)))

            # Check if it's divisible by 13 (Tony's lucky number)
            if number % 13 == 0:
                valid_houses.add(number)

    return sorted(list(valid_houses))

def main():
    n = int(input().strip())
    digits = list(map(int, input().strip().split()))

    pizza_houses = find_pizza_houses(digits)

    if pizza_houses:
        for house in pizza_houses:
            print(house)
    else:
        print("No pizza for anyone!")

if __name__ == "__main__":
    main()
```

25 NBA Dynamic Plus/Minus Tracking

15 points

Introduction

In basketball statistics, the *plus/minus* metric measures the impact of a player on the court. A player receives **positive points** when their team outscores the opponent while they are playing, and **negative points** when the opponent scores more during their time on the floor.

Real games also include substitutions – players rotate in and out of the court during the match. So you'll have to keep track of this as well.

Your mission is to step into the role of the coach of an NBA team to calculate **plus/minus value for each player** of your team.

Quick example:

Let's follow player A2 through the game.

- A2 started the match when score is 0-0.
- A2 is substituted when his team is winning 15-10.
- Their team scored 15 points and the opponent scored 10.
- He now has a *plus/minus* or differential of +5.

Later in the game:

- Same player enters the court again when the score is 40-30.
- He is substituted again when his team is losing 42-45.
- Their team scored 2 points, but the opponent scored 15.
- He has accumulated a *plus/minus* or differential of -13 this time.

He doesn't play anymore in this game, so his total *plus/minus* in the match is 5 - 13 = **-8**

Input

There is one set of input data, starting always with your team name ("Team CW").

Scores are cumulative throughout the game.

The input contains:

- One "START" line (**START A1 A2 A3 A4 A5**):
  - **START** declares the match is starting (score is 0-0).
  - **A1 A2 A3 A4 A5** are the players on court when match begins. There are always **5 players on the court** at all times. They are specified in natural sorting order (A1, A2, etc.). There are no more than 9 players (A1 to A9).

- **Zero or more** "SUB" lines, that report player substitutions (**SUB OUT A2 IN A6 SCORE 33 32**).
  - **SUB** declares a substitution.
  - **OUT A2** is the player leaving the court.
  - **IN A6** is the player entering the court.
  - **SCORE 33 32** is the **cumulative score** at that moment: Team CW has 33 points and the opponent has 32 points.

- One "END" line (**END SCORE 100 89**)
  - **END** declares the match is finished.
  - **SCORE 100 89** is the **final score**: Team CW has 100 points and the opponent has 89 points.

Output

The output must specify:

- a line the team name "Team CW". It's always the same name.
- a line for every player of Team CW with their respective differential or *plus/minus* score for the match, in natural sorting order (A1, A2, etc)

Players that have not played must not appear in the Output.

NOTES to remember:

- **Scores are cumulative** – they represent the total game score at that moment.
- There are always 5 players on the floor playing the game.
- SUBs never overlap (2 players won't be subbed at the same time).
- Same player can be substituted multiple times (leave and re-enter).

Example 1

Input

```
Team CW
START A1 A2 A3 A4 A5
SUB OUT A2 IN A6 SCORE 33 32
SUB OUT A4 IN A2 SCORE 70 62
SUB OUT A6 IN A4 SCORE 72 64
END SCORE 100 89
```

Output

```
Team CW
A1 11
A2 4
A3 11
A4 11
A5 11
A6 7
```

Example 2

Input

```
Team CW
START A1 A2 A3 A4 A5
SUB OUT A1 IN A6 SCORE 2 10
SUB OUT A6 IN A1 SCORE 6 18
SUB OUT A2 IN A7 SCORE 10 22
SUB OUT A7 IN A2 SCORE 14 26
SUB OUT A3 IN A8 SCORE 33 35
SUB OUT A8 IN A3 SCORE 46 38
SUB OUT A4 IN A9 SCORE 56 42
SUB OUT A9 IN A4 SCORE 97 80
END SCORE 110 89
```

Output

```
Team CW
A1 25
A2 21
A3 11
A4 18
A5 21
A6 -4
A7 0
A8 10
A9 3
```

Python

```python
# Read multiline input from user until blank line
def read_input_from_user():
    #print("Paste input for CW team\n")
    lines = []
    while True:
        try:
            line = input().strip()
        except EOFError:
            break
        if line == "":
            break
        lines.append(line)
    return lines

# Parse input and compute plus/minus
def solve(lines):
    idx = 1  # skip "Team CW" line

    # Parse START line: START A1 A2 A3 A4 A5
    parts = lines[idx].split()
    current_lineup = parts[1:6]  # five starting players
    idx += 1

    # Track players in order they appear
    players = current_lineup[:]
    pm = {p: 0 for p in players}

    # Track cumulative score (starts at 0-0)
    last_scoreA = 0
    last_scoreB = 0

    # Process SUB and END lines
    while idx < len(lines):
        parts = lines[idx].split()

        if parts[0] == "END":
            # END SCORE 100 89
            final_scoreA = int(parts[2])
            final_scoreB = int(parts[3])

            # Calculate +/- for final segment
            gained = (final_scoreA - last_scoreA) - (final_scoreB - last_scoreB)
            for p in current_lineup:
                pm[p] += gained
```

```
            break

        elif parts[0] == "SUB":
            # SUB OUT A2 IN A6 SCORE 33 32
            out_p = parts[2]
            in_p = parts[4]
            scoreA = int(parts[6])
            scoreB = int(parts[7])

            # Calculate +/- for this segment
            gained = (scoreA - last_scoreA) - (scoreB - last_scoreB)
            for p in current_lineup:
                pm[p] += gained

            # Apply substitution
            current_lineup[current_lineup.index(out_p)] = in_p

            # Add new player to roster if first time
            if in_p not in players:
                players.append(in_p)
                pm[in_p] = 0

            # Update score tracking
            last_scoreA = scoreA
            last_scoreB = scoreB

        idx += 1

    return players, pm

# Main program
def main():
    lines = read_input_from_user()

    # Solve
    roster, pm = solve(lines)

    # Output results
    print("Team CW")
    for p in roster:
        print(p, pm[p])

if __name__ == "__main__":
    main()
```

26 Broken Calculator (Interactive problem)

16 points

Introduction

You found an old calculator at a garage sale. The display is completely broken — you can't see any numbers! However, the calculator still works internally. It contains a secret number X (between 1 and N), and you can perform operations on it when the calculator is connected to your computer. Your mission: discover the secret number X using a limited number of operations!

The calculator supports three operations that will help you to discover X:

- ADD k → adds k to the current value
- SUB k → subtracts k from the current value
- CHECK → tells you if the current value is POSITIVE, NEGATIVE or ZERO

Important: Operations accumulate. The calculator remembers all operations you perform. For example, if X = 37 and you do SUB 50 then ADD 20, the current value becomes:
37 - 50 + 20 = 7.

Once you know the value of X ask the calculator whether the answer is correct with command: ANSWER X

Input

The calculator first sends two positive integers: **`N Q`**
where N is the upper bound (the secret X is in the range [1, N]) lesser or equal to 10000 and Q is the maximum number of operations allowed as $3 \times \lfloor \log_2(N) \rfloor + 3$

The calculator responds to **`CHECK`** operation with:

- **`POSITIVE`** if current value > 0
- **`NEGATIVE`** if current value < 0
- **`ZERO`** if current value = 0

The calculator responds to **`ANSWER k`** command with **`CORRECT`** or **`WRONG`**. After Q operations/tries the calculator will respond with **`LIMIT_EXCEEDED`**.

Output

The operations that can be requested are:

- To add to the current value: **`ADD k`**
- To subtract from the current value: **`SUB k`**
- To check the sign of the current value: **`CHECK`**

When you've determined the secret number, output: **`ANSWER x`**

Every time you want to send a command to the judge, you should do a flush. You can use `fflush(stdout)` **in C,** `cout.flush()` **in C++ and** `sys.stdout.flush()` **in Python.**

You can see a full example of an interactive problem in the Contestant's Guide.

Example

Calculator	Communication	Your program
100 24		
		SUB 51
		CHECK
NEGATIVE		
		ADD 25
		CHECK
NEGATIVE		
		ADD 13
		CHECK
POSITIVE		
		SUB 6
		CHECK
NEGATIVE		
		ADD 3
		CHECK
ZERO		
		ANSWER 16
CORRECT		

Python

```python
import sys


def flush_print(s):
    print(s, flush=True)


def read_response():
    return input().strip()


def check():
    """Check the current value sign."""
    flush_print("CHECK")
    response = read_response()
    if response == "LIMIT_EXCEEDED":
        sys.exit(1)
    return response


def sub(k):
    """Subtract k from current value."""
    flush_print(f"SUB {k}")


def add(k):
    """Add k to current value."""
    flush_print(f"ADD {k}")


def solve():
    # Read N and Q
    line = read_response()
    n, q = map(int, line.split())

    # Binary search for X
    # We track the cumulative offset: current_value = X - offset
    # To check if X >= mid, we need current_value - (mid - offset) >= 0
    # Which means we need to SUB (mid - offset) to make current = X - mid

    low, high = 1, n
    offset = 0  # Current value = X - offset

    while low < high:
        mid = (low + high + 1) // 2  # Upper mid to avoid infinite loop
```

```
        # We want current value to become X - mid
        # Current value is X - offset
        # So we need to subtract (mid - offset)
        delta = mid - offset

        if delta > 0:
            sub(delta)
        elif delta < 0:
            add(-delta)
        # If delta == 0, no operation needed, just check

        offset = mid  # Now current value = X - mid

        result = check()

        if result == "ZERO":
            # X - mid == 0, so X == mid, answer immediately!
            flush_print(f"ANSWER {mid}")
            return
        elif result == "POSITIVE":
            # X - mid > 0, so X > mid
            low = mid
        else:
            # X - mid < 0, so X < mid
            high = mid - 1

    # low == high == X
    flush_print(f"ANSWER {low}")


if __name__ == "__main__":
    solve()
```

27 Parallel 3D Printing

17 points

Introduction

You work for a company that owns several HP Multi-Jet Fusion 3D Printers, and you are in charge of distributing the parts among the printers to minimize the time it takes to print all the parts, while not spending too many resources to optimize the calculation. After some investigation, you have decided to go with a First-Fit Decreasing approach, which might not always be optimal, but can be calculated in polynomial time and is deterministic.

You are given a list of parts, each with a volume, and a list of printers, each with a maximum print capacity per batch. A printer processes parts in batches, and:

- The sum of volumes in a batch must not exceed the printer's capacity.
- Each batch takes exactly 1 time unit to print.
- Printers work in parallel.
- A printer may process multiple batches sequentially.

Your task is to assign parts to printers following a variation of the **First-Fit Decreasing (FFD) strategy** and then determine the total printing time.

Assignment Rules (Deterministic FFD Variant)

1. Sort the parts in decreasing order of volume.
2. For each part, in sorted order:

    a. Try to fit it into an existing open batch: For each printer in input order (1 to N), check the last open batch. If the part fits, place the part in the batch and continue to the next part.
    b. If the part does not fit in any existing batch, open a new batch on one of the printers.

The printer where the new batch is opened is chosen using the rule below.

New Batch Selection Rule

When a part needs a new batch, choose the printer with the smallest number of batches currently opened. If multiple printers have the same (minimum) number of batches, break ties by:

1. Printer with larger capacity
2. If still tied, smallest printer index (input order)

After all parts are assigned, each printer has a certain number of batches it must run sequentially.

Your task is to output the total printing time. Since printers work in parallel, the time required to finish the entire job is determined by the printer that has the most batches.

Input

The input consists of four lines
P
v1, v2, ... vP
N
c1, c2, ... cN

where:

- P: number of parts
- vi: volume of part i
- N: number of printers
- cj: capacity of printer j

Constraints

- $1 \le P \le 200\,000$
- $1 \le N \le 1000$
- $1 \le vi \le 10^9$
- $1 \le cj \le 10^{12}$

All parts are guaranteed to fit in at least one printer.

Output

Print a single integer representing the number of time units required to print all parts.

Example 1

Input

```
5
8 5 4 4 3
2
10 7
```

Output

```
2
```

Example 2

Input

```
6
5 5 1 9 8 8
2
10 10
```

Output

```
2
```

Example 3

Input

```
2
5 5
2
5 5
```

Output

```
1
```

Python

```python
import sys

def solve(data):
    # Read the data from stdin in case it is not provided by the external call
    if data == None:
        data = sys.stdin.read().strip().split()

    P = int(data[0])
    parts = []
    idx = 1
    for _ in range(P):
        volume = int(data[idx])
        parts.append(volume)
        idx += 1

    N = int(data[idx])
    idx += 1
    capacities = []
    for _ in range(N):
        capacity = int(data[idx])
        capacities.append(capacity)
        idx += 1

    # Sort parts decreasing for FFD
    parts.sort(reverse=True)

    # For each printer:
    # - batch_sums[i]: current last-batch filled volume (0 if last batch empty)
    # - batch_counts[i]: number of batches already opened for that printer (0
means no batch opened yet)
    batch_sums = [0] * N
    batch_counts = [0] * N

    for p in parts:
        placed = False

        # 1) Try to place into an existing (current) batch of each printer in
input order (FFD)
        for i in range(N):
            if batch_counts[i] > 0 and batch_sums[i] + p <= capacities[i]:
                batch_sums[i] += p
                placed = True
                #print(f"Placed {p} on printer {i}")
                break
```

```
        if placed:
            continue

        # 2) If no existing batch can fit the part, start a new batch.
        # Choose printer with:
        #   a) minimum batch_counts
        #   b) if tie, maximum capacity
        #   c) if tie, smallest index
        best_idx = None
        best_batch_count = None
        best_capacity = None

        for i in range(N):
            bc = batch_counts[i]
            cap = capacities[i]

            if best_idx is None:
                best_idx = i
                best_batch_count = bc
                best_capacity = cap
            else:
                # prefer smaller batch count
                if bc < best_batch_count:
                    best_idx = i
                    best_batch_count = bc
                    best_capacity = cap
                elif bc == best_batch_count:
                    # prefer larger capacity
                    if cap > best_capacity:
                        best_idx = i
                        best_capacity = cap
                    # if capacity equal, keep smaller index (best_idx is already
smaller by iteration order)

        # Start a new batch on chosen printer and place the part
        batch_counts[best_idx] += 1
        batch_sums[best_idx] = p

        #print(f"Placed {p} on printer {best_idx}")

    # All parts placed; time units = max batches across printers (0 for unused
printers)
    print(max(batch_counts))

if __name__ == "__main__":
    solve(None)
```

28 Largest Island

18 points

Introduction

The tropical storm has knocked out all power on Isla Nublar, and the dinosaurs are loose! As the emergency team scrambles to evacuate, rising floodwaters from the storm are swallowing the lower areas of the island. Dr. Grant and the survivors need to find the largest piece of high ground that the water cannot reach — a safe zone where they can wait for the rescue helicopter without becoming velociraptor snacks.

You are given a square grid representing a terrain with different elevations at each position. When water rises to a certain level, it will flood all areas at or below that water level. The water starts from the top-left corner (0,0) and spreads to adjacent cells (up, down, left, right) that are also at or below the water level.

Your task is to find the size of the largest island that is NOT flooded (not connected to the starting corner through water). An island is a group of adjacent cells (horizontally or vertically connected) that are not flooded. Note that an island can include cells below water level if they are protected by higher ground that prevents the water from reaching them.

Input

The first line contains a positive integer *n*, representing the size of the square grid (*n* x *n*).
The second line contains a positive integer *w*, representing the water level.
The next *n* lines each contain *n* space-separated integers representing the elevation at each position in the grid. Each row represents heights from left to right.

Output

A single integer representing the size (number of cells) of the largest island that is not flooded from the starting corner. This includes all connected non-flooded cells, even those below water level that are protected by higher terrain.

Example 1

Input

```
5
3
2 3 4 3 2
3 5 6 5 3
4 6 8 6 4
3 5 6 5 3
2 3 4 3 2
```

Output

```
22
```

Example 2

Input

```
4
2
1 1 3 3
1 1 3 3
4 4 1 1
4 4 1 1
```

Output

```
12
```

Example 3

Input

```
5
3
5 5 5 5 5
5 2 2 2 5
5 2 1 2 5
5 2 2 2 5
5 5 5 5 5
```

Output

```
25
```

Python

```python
def find_largest_island(size, water_level, grid):
    """
    Finds the largest island above water level that is not connected to water.
    Water starts flooding from position (0,0) if it's at or below water level.

    Args:
        size: Size of the square grid (n x n)
        water_level: The water level threshold
        grid: 2D list representing the heights at each position

    Returns:
        The size of the largest island above water
    """

    # Create a visited matrix to track flooded areas
    flooded = [[False] * size for _ in range(size)]

    # BFS to mark all flooded areas (connected to top-left corner through water)
    def flood_from_corner():
        if grid[0][0] > water_level:
            return  # Starting corner is above water, no flooding

        queue = [(0, 0)]
        flooded[0][0] = True

        while queue:
            r, c = queue.pop(0)

            # Check all 4 directions
            for dr, dc in [(0, 1), (1, 0), (0, -1), (-1, 0)]:
                nr, nc = r + dr, c + dc

                if (0 <= nr < size and 0 <= nc < size and
                    not flooded[nr][nc] and grid[nr][nc] <= water_level):
                    flooded[nr][nc] = True
                    queue.append((nr, nc))

    # Flood from the top-left corner
    flood_from_corner()

    # Find all islands (areas above water that are not flooded)
    visited = [[False] * size for _ in range(size)]

    def get_island_size(start_r, start_c):
```

```
        """DFS to calculate island size - includes protected areas below water
level"""
        stack = [(start_r, start_c)]
        visited[start_r][start_c] = True
        count = 0

        while stack:
            r, c = stack.pop()
            count += 1

            # Check all 4 directions
            for dr, dc in [(0, 1), (1, 0), (0, -1), (-1, 0)]:
                nr, nc = r + dr, c + dc

                # Include any cell that isn't flooded (regardless of height)
                if (0 <= nr < size and 0 <= nc < size and
                    not visited[nr][nc] and
                    not flooded[nr][nc]):
                    visited[nr][nc] = True
                    stack.append((nr, nc))

        return count

    # Find the largest island (start from any non-flooded cell)
    max_island_size = 0

    for i in range(size):
        for j in range(size):
            if not visited[i][j] and not flooded[i][j]:
                island_size = get_island_size(i, j)
                max_island_size = max(max_island_size, island_size)

    return max_island_size


def main():
    """Main function to get input and calculate largest island."""
    """
    print("=== Largest Island Above Water ===\n")

    # Example 1
    print("Example 1:")
    print("Grid size: 5")
    print("Water level: 3")
    print("Grid:")
    grid1 = [
        [2, 3, 4, 3, 2],
```

```
        [3, 5, 6, 5, 3],
        [4, 6, 8, 6, 4],
        [3, 5, 6, 5, 3],
        [2, 3, 4, 3, 2]
    ]
    for row in grid1:
        print(" ".join(map(str, row)))
    result1 = find_largest_island(5, 3, grid1)
    print(f"Largest island size: {result1}\n")

    # Example 2
    print("Example 2:")
    print("Grid size: 4")
    print("Water level: 2")
    print("Grid:")
    grid2 = [
        [1, 1, 3, 3],
        [1, 1, 3, 3],
        [4, 4, 1, 1],
        [4, 4, 1, 1]
    ]
    for row in grid2:
        print(" ".join(map(str, row)))
    result2 = find_largest_island(4, 2, grid2)
    print(f"Largest island size: {result2}\n")

    # Example 3
    print("Example 3:")
    print("Grid size: 5")
    print("Water level: 3")
    print("Grid:")
    grid3 = [
        [5, 5, 5, 5, 5],
        [5, 2, 2, 2, 5],
        [5, 2, 1, 2, 5],
        [5, 2, 2, 2, 5],
        [5, 5, 5, 5, 5]
    ]
    for row in grid3:
        print(" ".join(map(str, row)))
    result3 = find_largest_island(5, 3, grid3)
    print(f"Largest island size: {result3}\n")

    print("-" * 40)
    print("Now try it yourself!\n")
    """
    # Get the grid size
```

```
    size = int(input().strip())

    # Get the water level
    water_level = int(input().strip())

    # Read the grid
    grid = []
    for i in range(size):
        row = list(map(int, input().strip().split()))
        grid.append(row)

    # Calculate the largest island
    result = find_largest_island(size, water_level, grid)

    # Display the result
    print(result)


if __name__ == "__main__":
    main()
```

29 Painting Robot Assignment

22 points

Introduction

Carmen owns a car painting workshop that offers three colors: **White**, **Black**, and **Red**. During the day, cars arrive to be painted one by one. Carmen has two robots (A and B) and needs to decide which one paints each car.

The problem: when a robot switches to a different color, it must clean itself with a special cleaning chemical. This chemical is expensive and bad for the environment, so Carmen wants to use as little as possible.

Cleaning chemical needed to switch between colors (same cost both ways):

White ↔ **Black:** 25 mL
White ↔ **Red:** 40 mL
Black ↔ **Red:** 30 mL

Important rules:

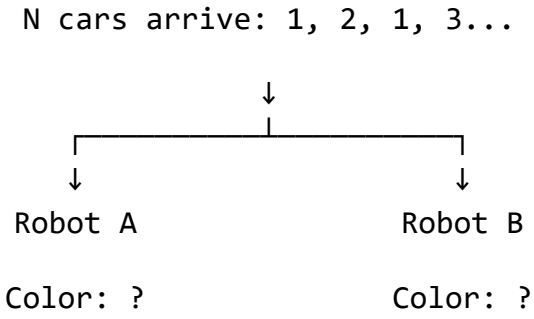
- The first car painted by each robot uses **no chemical** (the robot starts clean)
- If a robot paints the same color twice, it uses **no chemical**
- A robot only needs the cleaning chemical when it changes to a **different color**

Your mission: help Carmen assign cars to the two robots to **use the least amount of chemical possible!**

Input

The first line contains a single positive integer **N** (the number of cars for the day) where $1 \leq N \leq 100$.

The second line contains N integers, each representing the color for each car in arrival order. Colors are represented as: **1** = White, **2** = Black and **3** = Red.



Output

Your program should output a single line containing a single integer: the minimum total chemical used (in mL) from all color switches across both robots.

Example 1

Input

4

1 2 1 2

Output

0

Example 2

Input

6

1 1 1 2 2 2

Output

0

Example 3

Input

6

3 3 1 2 2 3

Output

25

Python

```python
def get_switch_cost(from_color, to_color):
    if from_color == to_color:
        return 0

    # Symmetric cost matrix
    costs = {
        (1, 2): 25,
        (1, 3): 40,
        (2, 3): 30,
    }

    key = tuple(sorted([from_color, to_color]))
    return costs.get(key, 0)


"""
State: dp[i][color_A][color_B] = minimum cost to assign first i cars
        - Robot A last used color_A, Robot B last used color_B
        - color_A and color_B are in {0, 1, 2, 3}
        - 0 means the robot has not been used yet, 1-3 are colors
"""
def solve_dp(cars):
    if len(cars) == 0:
        return 0

    n = len(cars)

    # Initialize DP states to infinity
    INF = float('inf')
    dp = [[[INF for _ in range(4)] for _ in range(4)] for _ in range(n + 1)]

    # No cars painted, no robots used
    dp[0][0][0] = 0

    for i in range(n):
        car_color = cars[i]

        for color_A in range(4):
            for color_B in range(4):
                if dp[i][color_A][color_B] == INF:
                    continue

                current_cost = dp[i][color_A][color_B]
```

```
                # If: assign car to Robot A
                if color_A == 0:
                    # Robot A hasn't been used yet
                    new_cost_A = current_cost + 0
                else:
                    new_cost_A = current_cost + get_switch_cost(color_A,
car_color)

                dp[i + 1][car_color][color_B] = min(dp[i +
1][car_color][color_B], new_cost_A)

                # If: assign car to Robot B
                if color_B == 0:
                    # Robot B hasn't been used yet
                    new_cost_B = current_cost + 0
                else:
                    new_cost_B = current_cost + get_switch_cost(color_B,
car_color)

                dp[i + 1][color_A][car_color] = min(dp[i +
1][color_A][car_color], new_cost_B)

    # Find min cost among all final states
    min_cost = INF
    for color_A in range(4):
        for color_B in range(4):
            min_cost = min(min_cost, dp[n][color_A][color_B])

    return min_cost


def main():
    n = int(input())
    cars = list(map(int, input().split()))

    result = solve_dp(cars)
    print(result)

if __name__ == "__main__":
    main()
```

30 Shiritori Chain

25 points

Introduction

You're coaching your friend Maya for the International Shiritori Championship! Shiritori is a Japanese word game where players take turns saying words, and each word must start with the last letter of previous word. For example:

apple → **elephant** → **tiger** → **rabbit** → **turtle** → ...

Maya wants to impress everyone with the longest possible word chain she can make. Given her vocabulary (a list of words she knows) and a starting word, help Maya find the length of the longest Shiritori chain she can create.

Rules:

1. Next word must start with the last letter of the previous word.
2. All words contain only lowercase letters (a-z).
3. No word can be used more than once in the chain.
4. The chain starts with the given starting word.
5. Only words from the vocabulary list can be used

Input

First line contains a starting word (this word is already in the vocabulary)
Second line has an integer N, the number of words in Maya's vocabulary
From line 3 to line N+2: One word per line (Maya's vocabulary, including the starting word)

Output

Print the results in the following format:

First line contains an integer representing the length in words of the longest chain.
Line 2 onwards: Each line contains one valid chain of maximum length, with words printed in the order they appear in the Shiritori sequence (not alphabetically), separated by spaces. Print all possible chains that achieve the maximum length, with each chain on its own line.

When there are multiple possible chains, sort them lexicographically: compare word by word, and order by the first word that differs between chains.

Example 1

Input

```
apple
5
apple
elephant
tiger
rabbit
turtle
```

Output

```
5
apple elephant tiger rabbit turtle
```

Example 2

Input

```
cat
6
cat
tiger
rat
turtle
eagle
dog
```

Output

```
5
cat tiger rat turtle eagle
```

Example 3

Input

```
zebra
3
zebra
apple
egg
```

Output

```
3
zebra apple egg
```

Example 4

Input

```
fox
1
fox
```

Output

```
1
fox
```

Example 5

Input

```
dog
7
dog
goat
tiger
rat
turtle
elephant
tuna
```

Output

```
7
dog goat tiger rat turtle elephant tuna
dog goat turtle elephant tiger rat tuna
```

Python

```python
def solve():
    # Read input
    start_word = input().strip().lower()
    n = int(input().strip())
    vocabulary = []
    for _ in range(n):
        vocabulary.append(input().strip().lower())

    # Build adjacency list for faster lookup
    # Map from first letter to list of words starting with that letter
    word_by_first_letter = {}
    for word in vocabulary:
        first_letter = word[0]
        if first_letter not in word_by_first_letter:
            word_by_first_letter[first_letter] = []
        word_by_first_letter[first_letter].append(word)

    # DFS with backtracking to find all longest chains
    def dfs(current_word, visited, current_chain, all_chains, max_length_ref):
        """
        Explores all paths and collects all chains with maximum length
        all_chains: list to collect all maximum-length chains
        max_length_ref: list with one element [max_length] to track current
maximum
        """
        current_length = len(current_chain)

        # Get the last letter of current word
        last_letter = current_word[-1]
        # Flag to check if we can extend further
        can_extend = False

        # Try all words that can follow (start with last_letter)
        if last_letter in word_by_first_letter:
            for next_word in word_by_first_letter[last_letter]:
                if next_word not in visited:
                    can_extend = True
                    # Mark as visited and explore
                    visited.add(next_word)
                    current_chain.append(next_word)

                    dfs(next_word, visited, current_chain, all_chains,
max_length_ref)
```

```
                    # Backtrack: remove from visited and chain
                    visited.remove(next_word)
                    current_chain.pop()

        # If we can't extend further, this is a terminal node
        if not can_extend:
            if current_length > max_length_ref[0]:
                # Found a longer chain, clear previous chains
                max_length_ref[0] = current_length
                all_chains.clear()
                all_chains.append(current_chain.copy())
            elif current_length == max_length_ref[0]:
                # Found another chain of same maximum length
                all_chains.append(current_chain.copy())

    # Start DFS from the starting word
    visited = {start_word}
    initial_chain = [start_word]
    all_chains = []
    max_length_ref = [0]

    dfs(start_word, visited, initial_chain, all_chains, max_length_ref)

    # Convert chains to strings (in Shiritori order, not alphabetically sorted)
    chain_strings = []
    for chain in all_chains:
        chain_strings.append(' '.join(chain))

    # Sort chains lexicographically (compare word by word)
    chain_strings.sort()

    # Remove duplicates (in case same chain found via different exploration
paths)
    unique_chains = []
    seen = set()
    for chain_str in chain_strings:
        if chain_str not in seen:
            seen.add(chain_str)
            unique_chains.append(chain_str)

    # Output the length first, then all unique chains
    print(max_length_ref[0])
    for chain_str in unique_chains:
        print(chain_str)

if __name__ == "__main__":
    solve()
```

31 The Counterfeit Coin (Interactive problem)

35 points

Introduction

You work in the Royal Mint's quality control department. A batch of N gold coins has been produced, but your sensors detected that exactly one coin is counterfeit. The counterfeit coin has a slightly different weight than the genuine coins - it could be either heavier or lighter, but you don't know which one.

Your only tool is a classic balance scale that compares the weight of two groups of coins. The scale tells you which side is heavier, or if both sides are equal. Your mission: identify the counterfeit coin AND determine whether it's heavier or lighter than the genuine coins.

The challenge? Can you code a program to find out the counterfeit coin in a limited number of weighings before your shift ends?

Important

- Coins are numbered from 1 to N
- Exactly one coin is counterfeit.
- The counterfeit coin is either strictly heavier OR strictly lighter (not equal)

Input

The first input contains a line sent by the judge's program `N K`, where `N` is a positive integer representing the number of coins (numbered 1 to N) and `K` is a positive integer as the maximum number of weighings allowed.
The rest of the inputs will come after your outputs.

If the output is a weigh request the judge will respond with one of `LEFT` when the left side is heavier, `RIGHT` when the right side is heavier or `EQUAL` if both sides weigh the same.

In case you exceed the number of weighings, the judge will respond with **`LIMIT_EXCEEDED`** and you lose.

When your program outputs an `ANSWER coin_number weight_type` the judge will respond with `CORRECT` or `WRONG`.

Output

To weigh coins, output a line in the format:

```
WEIGH left_coins : right_coins
```

where left_coins and right_coins are space-separated lists of coin numbers to place on each side of the balance

Rules:

- Both sides must have the same number of coins
- Each coin can appear at most once per weighing (no coin on both sides)
- You can leave some coins off the balance entirely

Once you have identified the counterfeit coin, output:

```
ANSWER coin_number weight_type
```

where `coin_number` is the number of the counterfeit coin (1 to N) and `weight_type` is either `H` (heavier) or `L` (lighter)

Every time you want to send a command to the judge, you should do a flush. You can use fflush(stdout) in C, cout.flush() in C++ and sys.stdout.flush() in Python.

You can see a full example of an interactive problem in the Contestant's Guide.

Example 1

Judge	Communication	Your program
3 2		
		WEIGH 1 : 2
LEFT		
		WEIGH 2 : 3
EQUAL		
		ANSWER 1 H
CORRECT		

Example 2

Judge	Communication	Your program
12 3		
		WEIGH 1 2 3 4 : 5 6 7 8
LEFT		
		WEIGH 4 5 6 : 7 8 9
RIGHT		
		WEIGH 5 : 6
LEFT		
		ANSWER 6 L
CORRECT		

Python

```python
#!/usr/bin/env python3
"""
Reference Solution for The Counterfeit Coin problem.

Strategy: Information-theoretic optimal ternary search.
Each weighing divides the 2N possibilities into 3 groups.
We select coins to weigh such that each outcome leaves ~1/3 of possibilities.
"""

import sys

def flush_print(s):
    print(s, flush=True)

def read_response():
    return input().strip()

def weigh(left, right):
    """Perform a weighing and return the result."""
    left_str = " ".join(map(str, left))
    right_str = " ".join(map(str, right))
    flush_print(f"WEIGH {left_str} : {right_str}")
    return read_response()

def get_outcome(left_set, right_set, coin, wtype):
    """What outcome would we see if (coin, wtype) is the counterfeit?
    Returns: 0=LEFT, 1=RIGHT, 2=EQUAL
    """
    if coin in left_set:
        return 0 if wtype == 'H' else 1
    elif coin in right_set:
        return 1 if wtype == 'H' else 0
    else:
        return 2

def partition_by_outcome(left_set, right_set, possibilities):
    """Partition possibilities by their outcome."""
    groups = [[], [], []]
    for poss in possibilities:
        outcome = get_outcome(left_set, right_set, poss[0], poss[1])
        groups[outcome].append(poss)
    return groups

def max_group_size(left_set, right_set, possibilities):
```

```
    """Return the size of the largest outcome group."""
    groups = partition_by_outcome(left_set, right_set, possibilities)
    return max(len(g) for g in groups)

def find_weighing(possibilities, genuine):
    """
    Find a good weighing using a direct approach:
    Try different subsets of suspects and evaluate which gives best partition.
    """
    suspects = sorted(set(coin for coin, _ in possibilities))
    genuine_list = sorted(genuine)
    n_poss = len(possibilities)
    target = (n_poss + 2) // 3

    best_left, best_right = None, None
    best_max = n_poss + 1

    # For small number of suspects, try all reasonable combinations
    n_suspects = len(suspects)

    # Optimal: about 1/3 of suspects on each side of scale
    ideal_size = max(1, n_suspects // 3)

    # Try sizes around the ideal
    for size in range(1, min(n_suspects // 2 + 2, n_suspects + 1)):
        # Put first 'size' suspects on left, next 'size' on right
        # This is a simple deterministic strategy

        for offset in range(n_suspects):
            left_coins = []
            right_coins = []

            idx = offset
            for _ in range(size):
                if idx < n_suspects:
                    left_coins.append(suspects[idx])
                    idx += 1
            for _ in range(size):
                if idx < n_suspects:
                    right_coins.append(suspects[idx])
                    idx += 1

            # Balance with genuine if needed
            temp_genuine = list(genuine_list)
            while len(left_coins) > len(right_coins) and temp_genuine:
                right_coins.append(temp_genuine.pop(0))
            while len(right_coins) > len(left_coins) and temp_genuine:
```

```
                left_coins.append(temp_genuine.pop(0))

            if len(left_coins) != len(right_coins) or len(left_coins) == 0:
                continue

            score = max_group_size(set(left_coins), set(right_coins),
possibilities)

            if score < best_max:
                best_max = score
                best_left = list(left_coins)
                best_right = list(right_coins)

                if best_max <= target:
                    return best_left, best_right

    if best_left is None:
        # Fallback: first suspect vs second or genuine
        if len(suspects) >= 2:
            best_left = [suspects[0]]
            best_right = [suspects[1]]
        elif len(suspects) == 1 and genuine_list:
            best_left = [suspects[0]]
            best_right = [genuine_list[0]]

    return best_left, best_right

def solve():
    line = read_response()
    n, k = map(int, line.split())

    possibilities = [(i, 'H') for i in range(1, n + 1)] + [(i, 'L') for i in
range(1, n + 1)]
    genuine = set()

    while len(possibilities) > 1:
        suspects = set(coin for coin, _ in possibilities)

        if len(suspects) == 1:
            coin = list(suspects)[0]
            types = [t for c, t in possibilities if c == coin]
            if len(types) == 1:
                break
            if genuine:
                gen = min(genuine)
                result = weigh([coin], [gen])
                if result == "LEFT":
```

```
                    possibilities = [(coin, 'H')]
                elif result == "RIGHT":
                    possibilities = [(coin, 'L')]
            break

        left, right = find_weighing(possibilities, genuine)

        if not left or not right:
            break

        left_set = set(left)
        right_set = set(right)

        result = weigh(left, right)

        if result == "EQUAL":
            genuine.update(left)
            genuine.update(right)
        else:
            on_scale = left_set | right_set
            off_scale = suspects - on_scale
            genuine.update(off_scale)

        result_idx = {"LEFT": 0, "RIGHT": 1, "EQUAL": 2}[result]
        groups = partition_by_outcome(left_set, right_set, possibilities)
        possibilities = groups[result_idx]

    if possibilities:
        coin, weight_type = possibilities[0]
        flush_print(f"ANSWER {coin} {weight_type}")
    else:
        flush_print("ANSWER 1 H")

if __name__ == "__main__":
    solve()
```